Herramientas para Computación de Altas Prestaciones

-Resolución de sistemas de Ecuaciones Lineales.

-Descomposición LU

Sistemas de Ec. Lineales

En esta práctica seguiremos trabajando con el archivo membrana; en particular, sobre la última versión, en la que los sistemas lineales se resolvían mediante archivos mex.

Vamos a utilizar la librería Lapack. Matlab dispone de sus librerías lapack/blas (son la versión de Intel: MKL). Sustituiremos nuestro código por llamadas a las subrutinas correspondientes.

Sistemas de Ec. Lineales

Entre los ejemplos disponibles en Matlab de programación con mex, está el ejemplo matrixDivide.c. Este es un mex que llama a la subrutina de Lapack DGESV, que sirve para resolver directamente sistemas de ecuaciones lineales.

En la carpeta de poliformat recursos\seminario 7 puedes encontrar el archivo matrixDivide64.c, que es similar al de Matlab pero adaptado para 64 bits. También puedes encontrar el archivo membrana_lapack.m y el creasis.m

Mex con Lapack/Blas (1)

Vamos a revisar los detalles relevantes del mex matrixDivide_64.c.

Cabecera

```
#if !defined(_WIN32)
#define dgesv dgesv_
#endif

#include "mex.h"
#include "lapack.h"
```

Mex con Lapack/Blas (2)

A continuación, obtención de punteros a argumentos de entrada y controles de errores

...

```
   A = mxGetPr(prhs[0]); /* pointer to first input matrix */
   B = mxGetPr(prhs[1]); /* pointer to second input matrix */
   /* dimensions of input matrices */
   m = mxGetM(prhs[0]);
   n = mxGetN(prhs[0]);
   p = mxGetN(prhs[1])
;...
```

Mex con Lapack/Blas (3)

Muy importante:

-Los argumentos de entrada de un mex (prhs) NO se pueden modificar, produce un error (“const”)

-Sin embargo, las subrutinas de Lapack y Blas modifican sus argumentos con mucha frecuencia; por ejemplo, DGESV modifica la matriz (calcula la descomposición LU sobreescribiendo la matriz de entrada)

-Por lo tanto: Es necesario tratar de forma especial los argumentos de entrada que vayan a ser modificados dentro de la subrutina Lapack: Crearemos una variable (matriz, vector,…) auxiliar de la misma dimensión y copiaremos el contenido del argumento de entrada en la variable auxiliar, que será la que se pase a la subrutina Lapack

Mex con Lapack/Blas (4)

La variable A apunta a la matriz de coeficientes del sistema de ecuaciones lineales, y la variable B apunta al vector lado derecho del sistema. Si las pasamos directamente a la subrutina Lapack, esta las modificará y se producirá un error. Para evitarlo, creamos variables auxiliares A2 y B2

```
Awork = mxCreateDoubleMatrix(m, n, mxREAL);
A2 = mxGetPr(Awork);
memcpy(A2, A, m*n*mxGetElementSize(prhs[0]));
```

```
plhs[0] = mxCreateDoubleMatrix(m, 1, mxREAL);
B2 = mxGetPr(plhs[0]);
memcpy(B2, B, m*mxGetElementSize(prhs[1]));
```

A la subrutina DGESV se le pasan las matrices A2 y B2

Mex con Lapack/Blas (5)

La subrutina DGESV necesita un vector de enteros para implementar la pivotación. Este vector se calcula al calcular la descomposición LU y se utiliza al resolver los sistemas triangulares. (Todo ello, dentro de la subrutina DGESV).

```
dims[0] = m;
mxPivot = mxCreateNumericArray(1, dims, mxINT64_CLASS, mxREAL);
iPivot = (mwSignedIndex*)mxGetData(mxPivot);
```

Paso de argumentos desde programas en C/C++ a subrutinas Fortran (Lapack/Blas)

http://es.mathworks.com/help/matlab/matlab_external/calling-lapack-and-blas-functions-from-mex-files.html#f44358

Mex con Lapack/Blas (6)

Al acabar DGESV:

- B2 (plhs[0]) apunta a la solución del sistema de ecuaciones lineales.
- A2 contiene la descomposición LU de la matriz original.
- Ipivot contiene el vector de pivotación utilizado.

Como en este caso A2 y Ipivot no se vuelven a utilizar, la memoria usada se libera:

```
mxDestroyArray(Awork);
mxDestroyArray(mxPivot);
```

Compilación de un mex que usa Lapack

Para compilar un mex con Blas o con Lapack, usaríamos estos comandos para identificar los paths correctos:

```
>>lapacklib =
fullfile(matlabroot,'extern','lib',computer('arch'),'microsoft',...
'libmwlapack.lib');
>>blaslib =
fullfile(matlabroot,'extern','lib',computer('arch'),'microsoft',...
'libmwblas.lib');
```

Si compilamos con Windows, podemos hacerlo así:

```
>>mex('-largeArrayDims', 'matrixDivide64.c', lapacklib)
```

Podemos añadir ***'-v'*** en la lista de argumentos para el modo "verbose".
'-largeArrayDims' es necesario para compilar con 64 bits.

Ejecución de membrana_lapack

Una vez compilado el mex matrixDivide64, podemos ejecutar el archivo membrana_lapack.

Podemos comprobar que su velocidad es similar o mejor a la del “slash” de Matlab. Y1=A\b.

Sin embargo, en la práctica anterior comprobamos que en este caso era conveniente resolver por separado la descomposición LU (que solo había que hacer una vez) y los sistemas de ecuaciones lineales triangulares (que había que hacer muchas veces.

Ejercicio:

Teneis que escribir dos archivos mex:

dgetrf_mex.c para calcular la descomposición LU de la matriz, usando la subrutina DGETRF. La llamada a este mex se pondrá fuera del bucle de membrana_lapack.

La llamada desde Matlab debería ser:

[M_lu,v_ind]=dgetrf_mex(A)

Donde A es la matriz de coeficientes, M_lu es la matriz que contiene la descomposición LU de A y v_ind es el vector de enteros que contiene los intercambios de filas

Ejercicio:

dgetrs_mex.c para resolver los dos sistemas triangulares conjuntamente, usando la subrutina DGETRS. La llamada a este mex debe ir dentro del bucle.

La llamada desde Matlab debería ser:

```
[y1]=dgetrs_mex(M_lu, b, v_ind)
```

Sistemas de Ec. Lineales

-Recordad que en las subrutinas mex, los parámetros de entrada no se deben modificar. Por otro lado, las subrutinas DGETRF y DGETRS utilizan sobreescritura:

DGETRF sobreescribe la matriz A con las matrices L y U

DGETRS sobreescribe el vector lado derecho b con la solución x.

Por lo tanto, en el gateway, hay que crear memoria y copiar el argumento de entrada que se va a sobreescribir, en la memoria creada tal como se hacía en MatrixDivide.c